The Software Quality Certification Triangle

Jeffrey Voas
Reliable Software Technologies

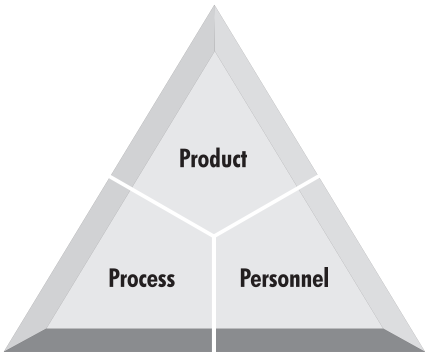
There are three distinct approaches to certifying the quality of software: accrediting personnel, certifying the development organization, and assessing the "goodness" of the software. These approaches, and hybrids thereof, are described, and criteria are given to determine which approach is best, depending on the software that needs to be certified.

Developing quality software is often considered elusive—it is more difficult to confidently know that you have developed good software than it is to build good software. In the physical sciences, the reverse is true—it is easier to measure the degree of perfection than it is to achieve perfection.

One reason why it is difficult to measure software quality stems from the many practical and theoretical deficiencies of software testing. For example, consider that to be 99 percent confident that a program has a probability of failure of less than one in 1 million, the software must be tested over 5 million times without observing a failure. Testing 5 million times requires that you have an oracle that is correct (an oracle is a person who knows or a program that knows what the correct software output is for all of the 5 million test cases). Rarely does a perfect oracle exist, and to create 5 million test cases would be intractable. And if you have the oracle and the test cases, there remains the impossible task of having to test using them.

Challenges such as these have made many in the software community decide that quality assessment of a software product is impractical. In addition to the traditional approach of assessing the "goodness" of the software, this has led to alternate approaches to software quality assessment. The two key competing approaches are process maturity assessment and accreditation of software professionals. The remainder of this article describes the pros and cons of these three approaches to predicting the quality of software.

Figure 1. *The software quality certification triangle.*



Accrediting Personnel

There are various ways to accredit, i.e., certify, personnel. The rigor with which personnel are certified depends on the criticality of the services that the person offers.

Professional licensing examinations, practical experience, and earned degrees are a few ways in which professionals can be accredited. For example, graduating from law school says something about a person's ability to practice law. It says less, however, than had the person also passed the bar. If this were not true, there would be no need for state bar examinations.

The intuition behind certifying "people skills" is simple; it should not be left up to the untrained consumer to be responsible to determine whether a candidate is qualified to perform the desired services. For example, how can Joe Public be expected to determine whether a dentist is qualified? Only if Joe Public were a dentist would he have any hope of making such a determination. By requiring dental school graduates to pass an examination prepared by dentists, the state takes the responsibility away from Joe Public. Further, if certified professionals do not live up to the expectations of their peers, they could be found liable and could lose their certification.

Like the older and more traditional professions of accounting, medicine, and law, the software industry is beginning to standardize the core principles each software professional should know. Microsoft claims that there are greater than 160,000 people who have become Microsoft certified as either product specialists, solution developers, trainers, or systems engineers [1]. This type of certification is "voluntary" (not required by any official governing organization) and expensive; however, the costs of certification can be recouped in the first year of working from the extra income the certificate enables. For example, it costs from $8,000 to $12,000 to become a Microsoft certified systems engineer (MCSE), and the total time to certify is approximately six months [1]. A person then can expect to make the same amount in additional income compared to a person who is not MCSE certified.

Just like doctors, lawyers, and certified public accountants, rumblings are also being heard concerning *mandatory* software engineering personnel certification. A vote by the Texas Board of Professional Engineers on Feb. 18, 1998 stated the board's intention to recognize software engineering as a legitimate engineering discipline and stated plans to license professional engineers in software engineering (a complete position statement from the Texas board can be found at http://www.main.org/peboard/softweng.htm). On June 17, 1998, the Texas board gave unanimous approval to all proposals in the statement. Beginning July 1999, the Texas board will license software engineers who can satisfy the following [2]:

- Possession of an engineering degree, a computer science degree, or some other high-level mathematics or

science degree that the board will evaluate for adequacy.
- At least 16 years of creditable experience performing engineering work (12 years for those who hold a degree approved by the Engineering Accreditation Commission of the Accreditation Board for Engineering Technology, Inc.).
- References from at least nine people, five of whom must be licensed engineers.
- Submission of documented credentials as required.

After the Texas board releases the professional software engineering examination in 1999, individuals with less experience will be allowed to obtain a Texas Professional Engineering license by passing the examination.

Assessing the Software Product

Generally, there are two approaches to product-based assessment of quality: white-box and black-box. White-box assessment techniques include activities such as collecting static code metrics or measuring the degree of coverage achieved during unit testing. Black-box techniques include reliability testing.

White-box and black-box techniques are not panaceas, however. For example, because reliability is based on logical correctness and the operational environment and not structural properties, it is unclear what relationship a code complexity metric has with the reliability of the software. Further, it is impossible to exhaustively test a simple program that reads in two 32-bit integers [4].

With today's push toward commercial-off-the-shelf (COTS) software, white-box certification techniques are normally not used by COTS consumers. However, white-box techniques may be applied by vendors if they wish to do so. Therefore, COTS consumers who are genuinely concerned about what lurks in the software they purchase must decompile back to source code to apply white-box analyses such as coverage testing or inspections.

Most COTS licenses deem this act a violation of the licensing agreement. Further, pending global legislation may weaken the ability of consumers to have such analysis done by independent corporations or consultants. In addition, a global treaty has been presented for U.S. approval entitled the World Intellectual Property Organization (WIPO) Treaty. The treaty includes language that makes it illegal to reverse engineer software to expose security vulnerabilities. The treaty will make it illegal for corporations and consulting services to conduct real-world testing of security software. Supposedly, research organizations will still be allowed to do so, however.

President Clinton has announced his intentions to sign the treaty, and it is expected to pass in the U.S. House of Representatives. The U.S. Senate has already passed the measure that deals with the treaty by a score of 99 to zero. The legislation is part of a global attempt to produce treaties that reduce the amount of copyright infringement on information technology. But the downside is that it disallows consumers the right to independently certify the security of the software they purchase (without the vendor's permission).

Certifying Processes

Because of the limitations associated with different forms of product assessment (testing as well as techniques such as formal verification), in the mid-1980s, the notion of "directly assessing software quality" was dismissed as implausible. This opened the door to ideas such as "process maturity assessment" and other indirect approaches. The most well-known process assessment model is the Software Engineering Institute Capability Maturity Model^{SM} for software. This model and other manufacturing-like standards rely on one premise—good processes deliver good software. This premise has also lead to government regulatory standards for software certification in avionics, medical devices, and electric power generation. The premise here is plausible. All developers have to do is score themselves using a pre-defined ranking scheme (for what is and is not good software development procedures), then apply that score to their software. For example, if development organization A is ranked higher than organization B, it is assumed that software from A has more quality than software from B. The problem is that good processes do not guarantee good software [6]. If performed properly, good processes merely increase the likelihood of producing quality products; if processes are not performed properly, the likelihood is reduced. However, given a fixed set of development processes, it is still possible that organization A, that improperly applies the set, produces better software than organization B, that properly applies the set. Furthermore, this does not account for issues related to which processes are "best." These facts, taken together, diminish the notion that process assessment will become a satisfactory substitute for product assessment. Ask yourself this: Would you buy a car without test driving it? Few would, but this is precisely what is done when process assessments are employed instead of product assessment. Process assessments are analogous to a car manufacturer that tells you what phases were undertaken during manufacture, which is no substitute for taking a test drive.

Software "Insurability"

I wil examine what role quality certification can play with respect to software insurability. Software insurability refers to the software-induced risk that an insurer is willing to take in exchange for an insurance premium. The insurer is not insuring the software but is instead insuring the object that the software controls. But before offering insurance for that object, the insurer must understand the worst-case scenarios that can result if the software is defective.

Consider that Swedish insurer Trugg-Hansa made the following exclusion effective May 1, 1998 in the general conditions of its business insurance policies.

> "The policy will not cover damage, cost, legal, or other liability caused directly or indirectly or connected to time-related disturbance in computer functionality."

This demonstrates the extreme, defensive posturing being seen as a result of the year 2000 problem. But of equal significance, it opens the door for nontime-related exclusions for other anomalous software behaviors. For example, exclusions might someday read as follows:

> "The policy will not cover damage, cost, legal, or other liability caused directly or indirectly or connected to disturbances in computer functionality."

Such a waiver enables an insurer to avoid responsibility for all computer-related problems. The onus is placed on consumers to know the quality of the computer systems they employ. Consumers now bear their own liability without access to an insurer to step in as their surrogate in case of a mishap. This represents a first in the software industry—insurers are so concerned about software failures that they have begun to include exclusions in their policies. When a situation such as this is coupled with the WIPO Treaty and the disregard for consumer protection that exists in the current version of the Uniform Commercial Code, Article 2B [3, 5], it is clear that the need for independent third-party certification concerning the processes, product, and personnel could not be greater.

Interestingly enough, a business has been formed to address the insurability problem—the Software Testing Assurance Corporation of Stamford, Conn. This company was founded in 1997 to provide independent certification. Their first certification offering will assess the testing processes used on year-2000-converted software. They currently certify most process assessments and a small portion of product assessments (their standard can be viewed at http://www.STACorp.com/draft/standard.htm). This independent certification is available only to corporations that seek business disruption insurance in the event their computer systems fail as a result of year 2000 software problems. The founding of this organization opens the door for additional software quality certification standards for information systems when business risks are directly tied to software quality and insurance protection is sought.

Summary

The hypothesis that certified personnel equates to higher quality software is easy to disprove. The hypothesis that a more mature process equates to higher quality software can also be easily debunked. Product assessment that studies the dynamic behavior of software is clearly the best approach to certifying software quality, but problems that relate to feasibility often reduce the ability to perform assessments with any degree or thoroughness.

The best approach is to create a variety of different certification schemes based on the different types of examinations or processes used from each of the three categories and the criticality of the software (flight control software vs. games). That is, aspects of each of these three broad approaches can be combined into a single standard. For example, knowing that an organization has a certain process maturity, the personnel who developed and tested the software were licensed, and the software received certain forms of quality assessment should result in greater confidence in the software's quality than if only one of these facts were known. The challenge, naturally, is how to quantify subjective characteristics such as personnel accreditation. Nonetheless, it is plausible to develop different software quality certification schemes that appropriately weigh different techniques within the three approaches with respect to the criticality of the software.

Acknowledgments

I thank Don Bagert for his efforts to keep me up to date on Texas' certification plans. ◆

About the Author

Jeffrey Voas is a co-founder of and chief scientist for Reliable Software Technologies and is currently the principal investigator on research initiatives for the Defense Advanced Research Projects Agency and the National Institute of Standards and Technology. He has published over 85 refereed journal and conference papers. He co-wrote *Software Assessment: Reliability, Safety, Testability* (John Wiley & Sons, 1995) and *Software Fault-Injection: Inoculating Programs Against Errors* (John Wiley & Sons, 1997). His current research interests include information security metrics, software dependability metrics, software liability and certification, software safety and testing, and information warfare tactics. He is a member of the Institute of Electrical and Electronics Engineers, and he holds a doctorate in computer science from the College of William & Mary.

Reliable Software Technologies
21515 Ridgetop Circle, Suite 250
Sterling, VA 20166
Voice: 703-404-9293
Fax: 703-404-9295
E-mail: jmvoas@rstcorp.com

References

1. Ayala, J., "Training the Microsoft Way," *Windows NT Magazine*, March 1998, pp. 122-129.
2. http://www.main.org/peboard/softweng.htm.
3. Kaner, C., "Article 2B is Fundamentally Unfair to Mass-Market Software Customers," submitted to the American Law Institute for its Article 2B review, October 1997.
4. Huang, J. C., "An Approach to Testing," *ACM Computing Surveys*, September 1975, pp. 113-128.
5. The American Law Institute and National Conference of Commissioners on Uniform Laws, Uniform Commercial Code, Article 2B (Draft), November 1997.
6. Voas, Jeffrey, "Can Clean Pipes Produce Dirty Water?" *IEEE Software*, July 1997, pp. 93-95.